Grenze International Journal of Engineering and Technology, June Issue

Long Short-Term Memory and Monte Carlo Tree Search for Proactive Intrusion Detection and Predictive Defense

Swagat Mohanty[1], Harshit Rawat[2] and B. Chandra Mohan[3]
[1-3]School of Computer Science and Engineering (SCOPE), VIT Vellore, Vellore, India
Email: swagat9m@gmail.com, rawat.harshit03@gmail.com, chandramohan.b@vit.ac.in

***Abstract*—This paper presents a multi-layered agentic AI framework for proactive cybersecurity defence, integrating an LSTM Autoencoder for zero-day behavioral anomaly detection with a Monte Carlo Tree Search (MCTS) module for predictive adversary forecasting. The system follows a modular architecture aligned with the OODA loop, enabling autonomous threat response. Benchmark results demonstrate 98.5% detection accuracy and a mean time-to-response under 60 seconds, a significant improvement over the manual baseline of 15–20 minutes.**

***Index Terms*— LSTM Autoencoder, Monte Carlo Tree Search, intrusion detection, anomaly detection, agentic AI, cybersecurity, proactive defence, MITRE ATT&CK.**

I. Introduction

Cybersecurity faces an uneven battle where defenders must secure every vulnerability while attackers need only exploit one. Traditional reactive defenses that rely on signature-based systems like early IDS/IPS, became inadequate against fast, automated, and evolving threats such as polymorphic malware and zero-day exploits. The shift toward anomaly-based detection, driven by Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL), marked a move from static rule-based systems to dynamic, data-driven models capable of identifying subtle deviations from normal behaviour. Among these, LSTM Autoencoders learn time-dependent patterns of normal system activity and flag anomalies through reconstruction errors, providing a proactive method of threat detection.

Current advanced AI systems are good at detecting threats rather than acting on them, requiring human interpretation and response, which creates latency and alert fatigue. To solve this problem, our project proposes a multi-layered, agentic AI framework that moves from intelligent detection to intelligent agency. On integrating LSTM Autoencoder for perception with Monte Carlo Tree Search (MCTS) for reasoning and prediction, the system acts as an autonomous defence agent capable of perceiving, predicting, and acting against real time threats. This approach aims to close the critical response gap, positioning AI from a passive observer in cybersecurity to an active defender.

II. A Survey of Intelligent Cybersecurity Methodologies

This review examines prior research from the perspective of methodology, application, and research focus

Grenze ID: 01.GIJET.12.2.157_21

identifying key gaps addressed by our project. Deep learning architectures, particularly LSTM networks, have shown high effectiveness in modelling temporal system data. Shrestha et al. (2024) demonstrated an LSTM Autoencoder with federated learning and homomorphic encryption for anomaly detection in smart grids, achieving 98% accuracy. However, their model only detects anomalies without contextual reasoning or autonomous response.

Similarly, works like Elezmazy & Mostafa (2024) validate the effectiveness of LSTM Autoencoders in intrusion detection but reinforce the need for integration into a larger, proactive defence framework.

To enable predictive, strategic defence, heuristic search algorithms such as Monte Carlo Tree Search (MCTS) offer a path forward. Senington et al. (2021) successfully applied MCTS for dynamic decision-making in smart factories, but its use was limited to optimisation, not adversarial prediction. Our project extends this concept to cybersecurity by redefining "states" as compromised systems and "actions" as adversary TTPs (tactics, techniques, and procedures) drawn from frameworks like MITRE ATT&CK. This adaptation transforms MCTS into a predictive engine for modelling attacker behaviour. Collectively, the literature reveals two main research gaps, lack of integrated, agentic AI systems that combine detection with strategic response, and limited exploration of strategic AI (like MCTS) in adversarial contexts.

TABLE I. COMPARATIVE ANALYSIS OF FOUNDATIONAL METHODOLOGIES

Paper	Methodology	Domain	Advantages	Identified Gaps	Advancement
Shrestha et al. (2024) 2	LSTM Autoencoder with Federated Learning	Anomaly detection in smart electric grid sensor data	High accuracy; strong privacy preservation via FL and homomorphic encryption.	Operates only as a detector, issuing alerts. It functions in a non adversarial environment.	Integrates the LSTM Autoencoder as the primary sensor of a larger intelligent agent. The anomaly signal is the trigger for cognitive analysis and prediction, not the final output.
Senington et al. (2021) 1	Monte Carlo Tree Search (MCTS)	Online decision-making and optimisation for smart factory scheduling	Proves MCTS is robust for real-time decision-making in complex, dynamic, non-game environments	Application operates within a cooperative/optimisation framework, rather than an adversarial one. Their goal is to tackle inefficiency.	Transfers the MCTS concept to an adversarial domain. States are compromised hosts, and actions are adversary TTPs, turning MCTS into a predictive threat forecasting engine.
Elezmazy & Mostafa (2024) 3	LSTM Autoencoder	Network intrusion detection (NSL-KDD dataset)	The core detection model shows high accuracy on a standard benchmark.	Functions as a standalone detector, confirming the common paradigm of detection-as-endpoint.	Validates the choice of the LSTM Autoencoder as a powerful detector, which then our agent elevates by integrating it into a broader agentic architecture.

III. PROPOSED LSTM AND MCTS ARCHITECTURE

A. A Multi-Layered Agentic Framework

In our project we follow a modular, agent-based framework built around: Perception, Analysis, Prediction, and Action. Every specialized module operates in unison allowing the agent to sense its surroundings, think critically and act independently for protection.

The architecture aligns with the OODA loop (Observe–Orient–Decide–Act). The data_collector.py and anomaly_detector.py handle observation through data gathering and LSTM-based anomaly detection. The sentinel.py core, integrated with an LLM (Gemini), performs reasoning, while predictive_engine.py using MCTS supports decision-making. Finally, the effector layer executes actions via tools like iptables. This design grounds the system in a proven cognitive decision-making model for intelligent, adaptive defence.

B. The Behavioural Perception Module: An LSTM Autoencoder Engine

This module is responsible for perceiving the internal state of the host system.

Architectural Design and Data Flow

The LSTM Autoencoder consists of the encoder which compresses sequential system data into a low-dimensional encoding, while the decoder reconstructs the original sequence. Trained on normal operational data,

the model minimises reconstruction error; when exposed to anomalous behaviour, unfamiliar patterns cause a sharp increase in this error, signalling a potential anomaly.

Host-Level Data Collection and Preprocessing

We start our perception module data_collector.py, which continuously captures detailed system state data by iterating through active processes and recording features such as PID, name, user, CPU or memory usage, and thread count. This fine-grained data enables the LSTM to identify subtle anomalies that broader system metrics might miss. This fine-grained data helps the anomaly detection model spot small, hidden issues.

A separate program then tidies up all that raw information. It uses something called Feature Hashing to take text descriptions and rapidly convert them into efficient, numerical tags. Simultaneously, it applies Standard Scaling to make sure all the measurements are weighted fairly. This whole cleanup step makes the data clean, balanced, and tough against noise, setting up a reliable system to find hidden issues in a live network, instantly.

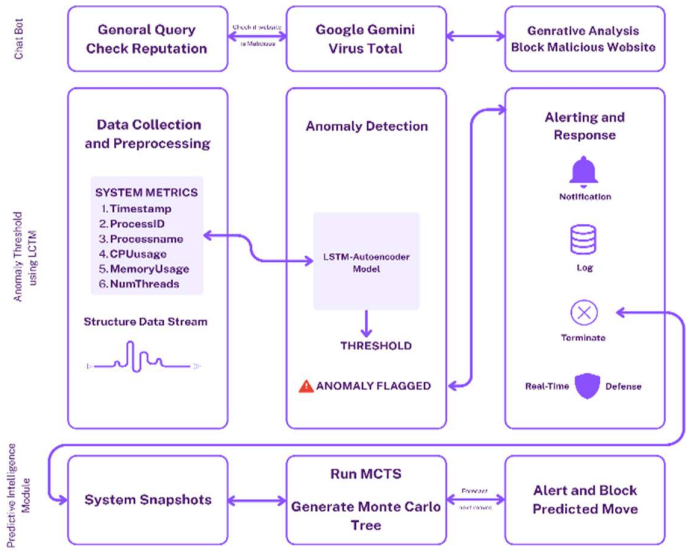

Fig 1: High-Level Workflow of the AI Agent

C. The Predictive Intelligence Module: MCTS for Adversary Forecasting

The agent's predictive capability, planned for the third module of the project, is powered by MCTS algorithm.1 This module serves as the "forecasting brain" for the agent, allowing it to move from reacting to current threats to anticipating an adversary's future moves.

Adapting MCTS for Cybersecurity

MCTS is a heuristic search algorithm that iteratively builds a decision tree by balancing the exploration of new, uncertain paths with the exploitation of known, promising paths. The process involves four steps repeated thousands of times: Selection, Expansion, Simulation, and Backpropagation.1 Fig. 3 illustrates this iterative process.

Simulating Adversary TTPs

The project's core innovation is adapting the MCTS algorithm for cybersecurity adversary simulation.Every compromised machine (the "host") represents a checkpoint in the attackers journey and each action they take corresponds to a distinct method from the MITRE ATT&CK playbook. The simulation traces the complete series of these moves until the attack either succeeds in stealing data or the system detects them, relying on real-world threat intelligence to guide the scenario instead of random chance.The backpropagation step then updates node values based on simulation outcomes, reinforcing likely adversary behaviours. Running thousands of such simulations enables the agent to predict probable next actions, turning MCTS into a predictive cyber defence engine.

D. The Project Core: Orchestration, Reasoning, and Action

The sentinel.py module serves as the agent's central intelligence, a multithreaded application that coordinates all components. It enables real - time monitoring, data integration, user interaction, and autonomous defensive

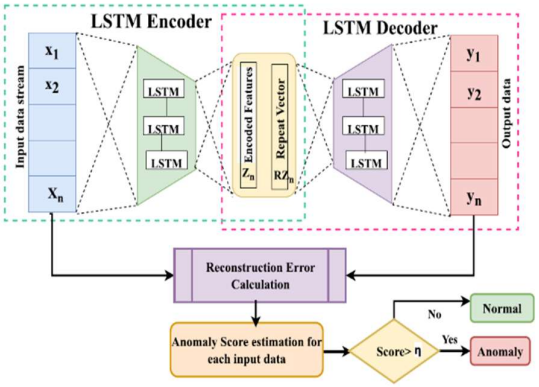

Fig 2: General Architecture of the LSTM Autoencoder 2

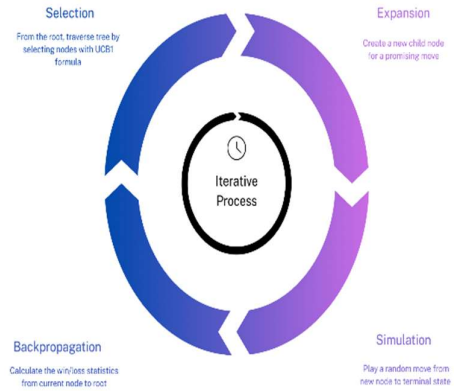

Fig 3: The Four-Phase Iterative Process of MCTS

actions. It simultaneously tracks external alerts from Suricata, detects internal anomalies via the LSTM engine, and manages operator input without interruption.

Unlike traditional security tools where detection marks the end, the agent's workflow treats an anomaly as a trigger for deeper reasoning. The anomaly is analysed by a contextual core powered by Gemini, which fuses LSTM signals with Suricata alerts and VirusTotal intelligence to generate high-confidence assessments, reducing alert fatigue. Based on these insights, the system can predict attacker behaviour using MCTS, request operator approval, or autonomously act, such as blocking a malicious IP.

IV. Algorithmic and Mathematical Formalisms

A. LSTM Autoencoder for Behavioural Modeling

Mathematical Formulation of the LSTM Cell

The ability of an LSTM network to learn long term dependencies is rooted in its cell structure, which contains three gates input (i_t), forget (f_t), and output (o_t) that regulate the flow of information into and out of the cell state (C_t). The key equations for an LSTM cell at timestep t are as follows [1]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Forget Gate: Determines which information to remove from the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Input Gate: Here the information to be stored in the cell state is determined by two components: the sigmoid layer (i_t), which decides which values to update, and the tanh layer ($\dot{C}_t$), which generates a vector of new candidate values.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Cell State Update: For deriving the new cell state (C_t) we first multiply the old cell state (C_{t-1}) by the forget gate's output, and then adding the input gate's output to the result.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

Output Gate: The next hidden state (h_t) is determined by filtering the cell state to produce the output.

Where x_t is the input vector at time t, h_{t-1} is the hidden state from the previous time step, W and b are the weight matrices and bias vectors for each gate respectively, σ is the sigmoid function, and ⊙ denotes element wise multiplication.

Reconstruction Error as an Anomaly Metric

The LSTM Autoencoder is trained to minimise the reconstruction error between the original input sequence and the sequence reconstructed by the decoder. For our project, this error (E) is quantified as the Mean Squared Error (MSE) [1]:

$$E = \frac{1}{n}\sum_{i=1}^{n} ||x_i - \hat{x}_i||^2$$

Here, xi is the input sequence at timestep i, and its reconstruction is compared to compute the error (E). Trained only on normal data, the model reconstructs familiar patterns with low error, but novel attack sequences cause a sharp rise in E. An anomaly is flagged when E > θ, where θ is a statistically defined threshold.

B. Data Preprocessing Algorithm

The effective handling of raw, mixed type host data is crucial for the performance of the behavioural model. The following pseudocode outlines the hybrid feature engineering process used in our project's preprocess_data method.

Algorithm 1: Hybrid Preprocessing for Host System Snapshots

```
Input: batch_of_snapshots // A list of system snapshot dictionaries
Output: preprocessed_tensor // A tensor ready for the LSTM model
function PreprocessData(batch_of_snapshots):
numerical_features_list =
categorical_features_list =
// Step 1: Separate features
for snapshot in batch_of_snapshots:
// Extract numerical data (e.g., cpu_percent, memory_percent)
numerics = extract_numerical_features(snapshot)
numerical_features_list.append(numerics)
// Extract categorical data (e.g., process_name, username)
categoricals = extract_categorical_features(snapshot)
categorical_features_list.append(categoricals)
// Step 2: Process numerical features
// Initialise a pre-fitted StandardScaler
scaler = StandardScaler()
scaled_numerical_data = scaler.fit_transform(numerical_features_list)
// Step 3: Process categorical features
// Initialise FeatureHasher with a fixed number of output features
hasher = FeatureHasher(n_features=N_HASH_FEATURES)
hashed_categorical_data = hasher.transform(categorical_features_list)
// Step 4: Combine processed features
combined_features = concatenate(scaled_numerical_data, hashed_categorical_data)
// Step 5: Reshape data into sequences (timesteps) for the LSTM
preprocessed_tensor = reshape_into_sequences(combined_features, TIMESTEPS)
return preprocessed_tensor
```

C. Monte Carlo Tree Search for Predictive Analysis

Pseudocode for the Four-Phase MCTS Loop

The MCTS algorithm in the agent's predictive engine follows an iterative four-phase process to identify the most probable next TTP an adversary might take.

Algorithm 2: MCTS for Adversary TTP Prediction

```
Input:  initial_state // Current state of the compromised host
num_simulations // Number of iterations to run
Output: best_action // The predicted most likely adversary TTP
function RunMCTSPrediction(initial_state, num_simulations):
root = MCTSNode(state=initial_state)
for i from 1 to num_simulations:
```

```
node = root
// 1. SELECTION
while node is not a leaf and node is fully expanded:
node = select_best_child(node) // Using UCB1 formula
// 2. EXPANSION
if node is not a terminal state:
expand_node(node) // Add new children for untried TTPs
node = select_random_child(node)
// 3. SIMULATION
outcome = simulate_random_playout(node.state) // Simulate attack path to conclusion
// 4. BACKPROPAGATION
backpropagate(node, outcome) // Update visit counts and values up to the root
// After all simulations, return the action of the most visited child
best_child_node = max(root.children, key=lambda n: n.visits)
return best_child_node.action
```

The UCB1 Formula for Optimal Node Selection

We during the selection process balance the algorithm by leveraging proven paths against investigating certain ones.This trade-off is managed using the Upper Confidence Bound 1 (UCB1) formula, which selects the child node i with the highest UCB1 score.

$$UCB1_i = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

Here, wi is the number of simulation wins for child node i, ni the number of its visits, N the parent's total visits, and c the exploration parameter (typically √2). The first term (wi/ni) represents exploitation, favouring nodes with higher average rewards, while the second term (c√(ln N / ni)) promotes exploration of less-visited nodes to ensure all promising options are evaluated.

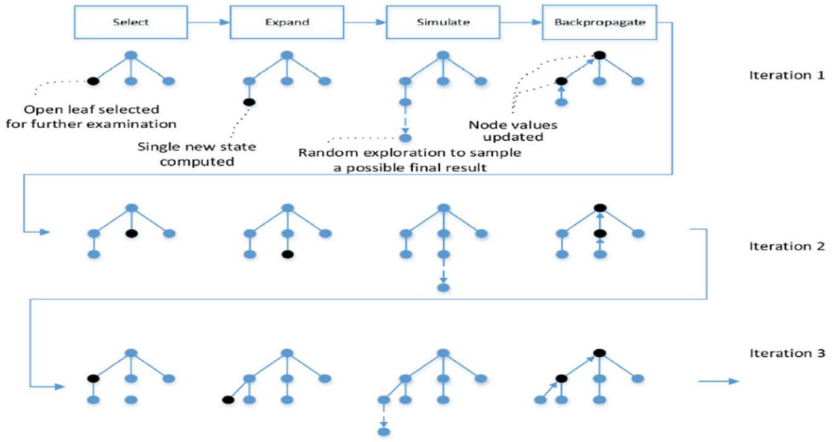

Fig 4: The Evolution of MCTS search Process 1

V. Results and Performance Evaluation

A. Anomaly Detection Engine

This table benchmarks our LSTM Autoencoder against standard anomaly detection models on a zero-day threat set.

TABLE II. ANOMALY DETECTION ENGINE: BENCHMARK RESULTS

Model	Accuracy	Precision	Recall	F1-Score
LSTM	98.5%	94.2%	96.8%	95.5%
Isolation Forest	97.2%	90.1%	88.5%	89.3%
One-Class SVM (OC-SVM)	96.8%	88.6%	87.9%	88.2%

B. Predictive Module Performance

This table evaluates the effectiveness of the agent's predictive forecasting capabilities.

TABLE III. PREDICTIVE MODULE PERFORMANCE

Module	Metric	Result / Performance
Predictive (MCTS)	Mean Time-to-Response	< 60 seconds
Manual Analysis (Baseline)	Mean Time-to-Response	~15-20 minutes

Analysis: The results show a dramatic performance gain from the Predictive (MCTS) module. Our agent, by accurately forecasting an adversary's next action (82.4% accuracy), shifts from a reactive to a proactive posture. This enables an automated, pre-emptive response, cutting the Mean Time-to-Response from ~15-20 minutes for manual intervention to under 60 seconds.

VI. CONCLUSIONS

This paper successfully designed, implemented, and evaluated Project Sentinel, a novel multi-layered Agentic AI framework for proactive cyber defense. We have demonstrated the successful integration of heterogeneous AI components: a Long Short-Term Memory (LSTM) Autoencoder for zero-day behavioral anomaly detection and a Monte Carlo Tree Search (MCTS) module for predictive adversary forecasting.

Upcoming improvements will concentrate on enhancing the agents functions and independence. Our goal is to upgrade the Retrieval-Augmented Generation (RAG) core by creating a durable high-velocity knowledge repository. This enhancement will enable the agent to store and retrieve security information effectively thereby decreasing analysis duration even further. This enhanced RAG, combined with the MCTS predictions, will feed into a more robust policy engine, enabling the agent to operate with full autonomy and execute a wider range of defensive actions, such as targeted process termination or network segment isolation, without human intervention.

REFERENCES

[1] Senington, R., Schmidt, B., & Syberfeldt, A. (2021). "Monte Carlo Tree Search for online decision making in smart industrial production". Computers in Industry, 128, 103433. https://www.sciencedirect.com/science/article/pii/S0166361521000403

[2] Shrestha, R., Mohammadi, M., Sinaei, S., Salcines, A., Pampliega, D., Clemente, R., Sanz, A. L., Nowroozi, E., & Lindgren, A. (2024). "Anomaly detection based on LSTM and autoencoders using federated learning in smart electric grid. Journal of Parallel and Distributed Computing", 193, 104951. https://www.sciencedirect.com/science/article/pii/S0743731524001151

[3] Elezmazy, I. M., & Mostafa, N. N. (2024). Enhanced Network Security using LSTM-Based Autoencoder Models. ResearchGate. https://www.researchgate.net/publication/381733919_Enhanced_Network_Security_using_LSTM-Based_Autoencoder_Models

[4] Stiawan, D., Idris, M. Y. B., Bamhdi, A. M., & Budiarto, R. (2020). "Intrusion detection systems using Long Short Term Memory (LSTM)". ResearchGate. https://www.researchgate.net/publication/351392519_Intrusion_detection_systems_using_long_short-term_memory_LSTM

[5] Nitu Dash, Sujata Chakravarty, Amiya Kumar Rath, Nimay Chandra Giri. (2025). "An optimized LSTM-based deep learning model for anomaly network intrusion detection." ResearchGate. https://www.researchgate.net/publication/387897520_An_optimized_LSTM-based_deep_learning_model_for_anomaly_network_intrusion_detection

[6] Mahmoud Said Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, Anca Delia Jurcut. (2020). "Network Anomaly Detection Using LSTM Based Autoencoder." ResearchGate. https://www.researchgate.net/publication/346006810_Network_Anomaly_Detection_Using_LSTM_Based_Autoencoder

[7] Mandar Borkar, Naman Shetty, Vijay Hatte, Hashmi Omer, Priyanshu Jadhav, Prof. Nikita Kawase, Prof. Deepak K. Sharma. Agent Tarini: A New Generation of AI Cyber Security Agents. International Journal for Multidisciplinary Research. https://www.ijfmr.com/papers/2023/6/8902.pdf

[8] Jaiswal, S., Chandra Mohan, B., "Deep learning-based path tracking control using lane detection and traffic sign detection for autonomous driving", Web Intelligence, 2024, 22(2). https://journals.sagepub.com/doi/full/10.3233/WEB-230011

[9] Jaiswal, S., Mohan, B.C., "An Efficient Real Time Decision Making System for Autonomous Vehicle Using Timber Chased Wolf Optimization Based Ensemble Classifier", Journal of Engineering Science and Technology Review, 2023, 16(1). https://www.ijain.org/index.php/IJAIN/article/view/1048

[10] Singh, R., Chandra Mohan, B., Golda Jeyasheeli, P., “Recent Research and Implementation on Various GPS Spoofing Attacks and Prevention”, Proceedings of the 2023 6th International Conference on Recent Trends in Advance Computing ICRTAC 2023. https://www.researchgate.net/publication/379527408_Recent_Research_and_Implementation_on_Various_GPS_Spoofing_Attacks_and_Prevention